ESTRUTURAS DE DADOS E ALGORITMOS

Departamento de Engenharia Eletrotécnica e de Computadores
Faculdade de Ciências e Tecnologia da Universidade de Coimbra

FICHA 9
PILHAS E FILAS

9.1. Objetivos

Objetivos que o aluno é suposto atingir com esta ficha:
- Compreender o conceito de pilha e o conceito de fila;
- Conhecer a sua utilidade em programação;
- Saber manipular de forma adequada as pilhas e as filas, podendo facilmente implementar/alterar funções que manipulam estes tipos particulares de listas ligadas.

9.2. Pilhas

Uma das estruturas ligadas de dados mais simples de manipular é a pilha. É uma das estruturas de dados mais utilizadas em programação, sendo inclusive usada na maioria das arquiteturas modernas de computadores. O princípio fundamental de uma pilha é que qualquer acesso aos seus elementos é feito através do elemento que está guardado no topo da pilha. Quando um elemento novo é inserido na pilha, passa a ser o elemento do topo e passa a ser também o único elemento que pode ser removido de imediato da pilha. Isto faz com que os elementos da pilha sejam retirados pela ordem inversa daquela com que foram inseridos: "o primeiro que sai é o último que entrou"; a sigla LIFO – "*last in, first out*" – é usada para descrever este tipo de acesso à estrutura de dados.

Para se entender o funcionamento de uma pilha, pode-se fazer uma analogia com uma pilha de pratos: ao adicionar-se um prato à pilha, este é colocado no topo; ao retirar-se um prato da pilha, é retirado o prato situado no topo da pilha. Uma estrutura de dados do tipo pilha funciona de maneira análoga.

Existem duas operações básicas para manipular uma estrutura de dados do tipo pilha: a operação para inserir um novo elemento no topo da pilha e a operação para retirar um elemento do topo da pilha. É comum designar estas duas operações através dos termos em inglês "*push*" (colocar na pilha) e "*pop*" (retirar da pilha).

Pode-se implementar uma pilha de números inteiros através das seguintes classes de objetos:

```
class CPilhaInteiros;

class CNoPilha{
    int dados;
    CNoPilha *proximo;
    friend class CPilhaInteiros; // esta classe acede aos membros private
};
```

```
class CPilhaInteiros{
    CNoPilha *topo;  // ponteiro para o elemento no topo da pilha
  public:
    CPilhaInteiros(void);
    ~CPilhaInteiros(void);

    void push(const int item);
    bool pop(int &item);
    void escrevePilha(void) const;
    bool pilhaVazia() const { return (topo==nullptr);}
};
```

A classe `CPilhaInteiros` tem um único atributo que armazena um ponteiro para o elemento que está no topo da pilha. São definidos alguns métodos que permitem manipular a pilha.

O primeiro dos métodos indicados é o construtor da classe que apenas inicializa o atributo `topo` com o valor `nullptr`, indicando assim que a pilha está inicialmente vazia.

```
CPilhaInteiros::CPilhaInteiros() {
      topo = nullptr;
}
```

É importante também definir um destrutor porque é necessário libertar a memória associada a cada um dos elementos (nós) da pilha, antes de se destruir o objeto `CPilhaInteiros`. Para tal, percorre-se a pilha utilizando um ponteiro auxiliar que aponta para o elemento a seguir àquele que se pretende apagar na iteração atual. Desta forma, é possível destruir o elemento atual sem perder uma ligação aos elementos seguintes na pilha (apontados por `proximo`).

```
CPilhaInteiros::~CPilhaInteiros() {
      CNoPilha *seguinte;

      while (topo != nullptr) {
            seguinte = topo->proximo;
            delete topo;
            topo = seguinte;
      }
}
```

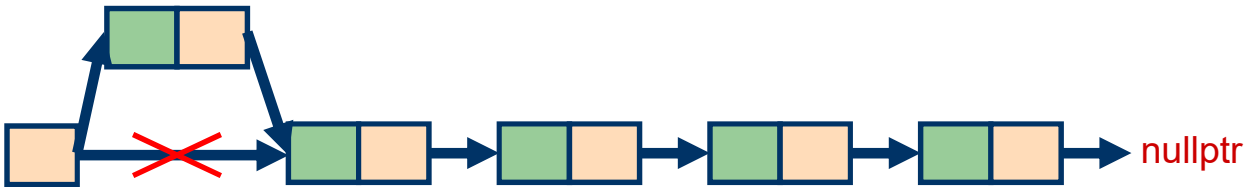
A definição do método `void push(const int)`, utilizado para inserir um novo inteiro no topo da pilha, é a seguinte:

```
void CPilhaInteiros::push(const int item) {
	CNoPilha *novo = new CNoPilha;

	novo->dados = item;
	novo->proximo = topo;

	topo = novo;
}
```

A figura a seguir apresenta de forma esquemática o método utilizado para inserir um novo elemento no topo da pilha:

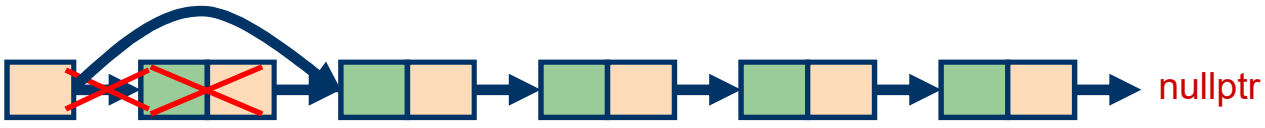


A definição do método `bool pop(int&)`, utilizado para retirar um elemento do topo da pilha, é a seguinte:

```
bool CPilhaInteiros::pop(int &item) {
	if (topo == nullptr) return false;	// a pilha está vazia!

	CNoPilha *aux = topo;
	topo = topo->proximo;			// atualiza o topo da pilha
	item = aux->dados;
	delete aux;
	return true;
}
```

A figura a seguir apresenta de forma esquemática o método utilizado para retirar um dado elemento do topo da pilha:



Finalmente, apresenta-se um método que permite escrever no ecrã os valores armazenados na pilha de inteiros:

```
void CPilhaInteiros::escrevePilha(void) const {
      if (topo == nullptr)
          cout << "Pilha Vazia... " << endl;
      else {
          CNoPilha *aux = Topo;

          while(aux != nullptr) {
              cout << aux->dados << endl;
              aux = aux->proximo;
          }
      }
}
```

9.3. Filas

Outra estrutura de dados que é muito utilizada em computação é a fila. Numa fila, o acesso aos seus elementos segue uma regra diferente: enquanto na pilha "o último que entra é o primeiro que sai", na fila "o primeiro que entra é o primeiro que sai"; a sigla FIFO – "*first in, first out*" – é usada para descrever esta estratégia. A ideia fundamental da fila é que só se pode inserir novos elementos no final da fila e retirar elementos do início da fila.

A estrutura de uma fila é uma analogia natural com o conceito de fila que usamos no nosso dia-a-dia: o primeiro a entrar numa fila é o primeiro a ser atendido (a sair da fila). Um exemplo de utilização em computação é a implementação de uma fila de impressão. Se uma impressora é compartilhada por várias máquinas, deve-se adotar estratégias para determinar que documento será impresso em primeiro lugar. A estratégia mais simples é tratar todas as requisições com a mesma prioridade e imprimir os documentos na ordem pela qual foram submetidos – o primeiro a ser submetido é o primeiro a ser impresso.

Para implementar uma fila, é necessário inserir novos elementos numa das extremidades, o fim, e retirar elementos da outra extremidade, o início. Para tal, é necessário definir dois ponteiros para referenciar cada um destes dois elementos.

Pode-se implementar uma fila de inteiros através da seguinte definição de classe:

```
class CFilaInteiros;

class CNoFila{
      int dados;
      CNoFila *proximo;

      friend class CFilaInteiros; // esta classe acede aos membros private
};
```

```
class CFilaInteiros {
    CNoFila *inicio; // ponteiro para o início da Fila
    CNoFila *fim;    // ponteiro para o fim da Fila
  public:
    CFilaInteiros(void);
    ~CFilaInteiros(void);

    void insereNaFila(const int item);
    bool retiraDaFila(int &item);
    void escreveFila(void) const;
    bool filaVazia(void) const { return (inicio == nullptr); }
};
```

A classe `CFilaInteiros` tem como atributos dois ponteiros. O primeiro serve para referenciar o elemento que está no início da fila e o segundo o último elemento da fila. A classe define ainda alguns métodos que permitem manipular a fila. O primeiro dos métodos indicados é o construtor que apenas inicializa com o valor `nullptr` os dois ponteiros para as extremidades da fila, indicando assim que a fila está inicialmente vazia.

```
CFilaInteiros::CFilaInteiros() {
      inicio = nullptr;
      fim = nullptr;
}
```

Mais uma vez, é importante libertar a memória associada a cada um dos nós que constituem a fila quando esta é destruída. Para tal, define-se um destrutor que percorre a fila utilizando um ponteiro auxiliar que aponta para o próximo elemento àquele que se pretende apagar na iteração atual. Desta forma, é possível destruir o elemento atual sem perder uma ligação aos elementos seguintes na fila.

```
CFilaInteiros::~CFilaInteiros (){
      CNoFila *seguinte;

      while (inicio != nullptr){
            seguinte = inicio->proximo;
            delete inicio;
            inicio = seguinte;
      }
      fim = nullptr;
}
```

A definição do método `void InsereNaFila(const int)`, utilizado para inserir um novo elemento no fim da fila, é a seguinte:

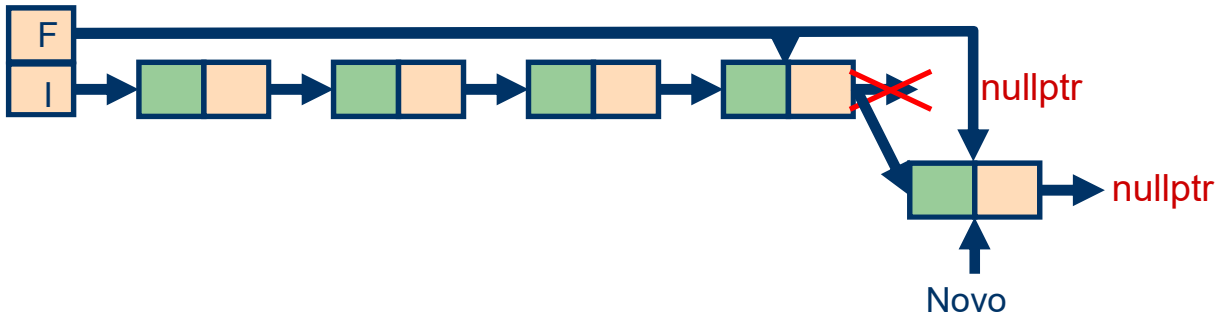
```
void CFilaInteiros::insereNaFila(const int item) {

	CNoFila *novo = new CNoFila;

	novo->dados = item;
	novo->proximo = nullptr;

	if (inicio == nullptr) inicio = novo;
	else fim->proximo = novo;
	fim = novo;
}
```

A figura a seguir apresenta de forma esquemática o método utilizado para inserir um novo elemento no fim da fila:



A definição do método `bool retiraDaFila(int&)` é a seguinte:

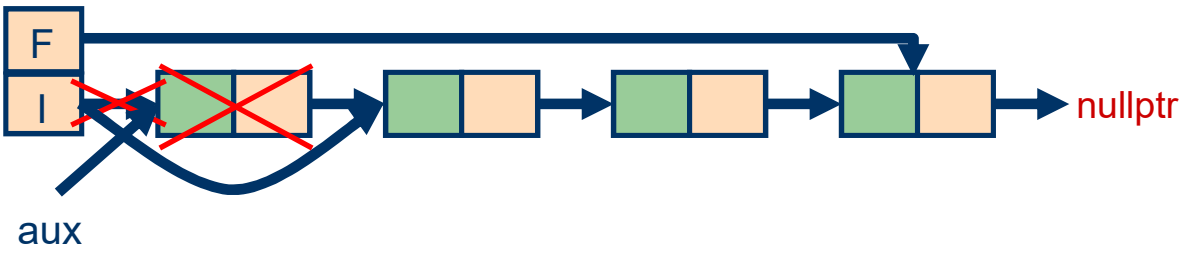
```
bool CFilaInteiros::retiraDaFila(int &item) {
	if (inicio==nullptr)  return false;

	item = inicio->dados;

	CNoFila *aux = inicio;		// ponteiro auxiliar para nó a eliminar

	inicio = inicio->proximo;	// atualiza ponteiro inicio da fila
	if (inicio == nullptr) fim = nullptr;  // se a fila ficar vazia
	delete aux;
	return true;
}
```

A figura a seguir apresenta de forma esquemática o método utilizado para retirar o elemento no início da fila:



Finalmente, apresenta-se um último método que permite escrever no ecrã os valores armazenados na fila de inteiros:

```
void CFilaInteiros::escreveFila(void) const {
	if (inicio == nullptr) {
		cout << "Fila vazia..." << endl;
		return;
	}

	for (CNoFila *aux=inicio; aux!=nullptr; aux=aux->proximo) {
		cout << aux->dados << endl;
	}
}
```

9.4. Exercícios sugeridos

Com o conjunto de exercícios apresentados a seguir, pretende-se que o aluno consolide os seus conhecimentos relativos às pilhas e filas. Para permitir uma melhor orientação do estudo, cada exercício foi classificado de acordo com o seu nível de dificuldade. Aconselha-se o aluno a tentar resolver em casa os exercícios não realizados durante as aulas práticas. Os problemas assumem a definição das classes `CPilhaInteiros` e `CFilaInteiros` apresentadas anteriormente. O código fonte da declaração destas classes e da definição de alguns dos seus métodos foi disponibilizado no InforEstudante juntamente com este enunciado.

Problema 9.1 – Fácil

Implemente uma função que, utilizando uma pilha, inverta uma *string* passada por parâmetro à função.

Problema 9.2 – Fácil

a) Defina um método para substituir o último elemento de uma fila de números inteiros pelo maior de todos os seus elementos. Note que o método pretendido só altera o valor do último elemento da fila, não alterando mais nenhum elemento.
b) Defina um método que troca o último elemento de uma fila de números inteiros com o maior de todos os seus elementos. Só precisa de trocar os dados, não sendo necessário mover os nós da fila.

Problema 9.3 – Fácil

Defina os seguintes métodos na classe `CPilhaInteiros`:

- `int numElementos(void)` – devolve o número de elementos contidos na pilha.
- `void trocaTopo(int valor)` – substitui o valor no topo da pilha pelo valor passado como parâmetro.

Problema 9.4 – Médio

Defina o método `bool CFilaInteiros::passaAfrente(int N)` que passa o último elemento na fila para a posição `N` a contar do início da fila. Se o valor de `N` é superior ao número de elementos contidos na fila o método devolve `false`, caso contrário devolve `true`.

Problema 9.5 – Médio

Defina um novo construtor `CPilhaInteiros(const CFilaInteiros &fila)` que constrói uma pilha de números inteiros a partir dos números inteiros contidos na fila passada como parâmetro.

Problema 9.6 – Médio

Implemente uma função booleana que, com o auxílio de uma estrutura de dados do tipo pilha, faça a análise de expressões matemáticas com o objetivo de verificar o balanceamento dos seus parêntesis retos e curvos. A função deve detetar erros de falta de parêntesis, como por exemplo, ***((a+b)***, e erros de parêntesis trocados, como por exemplo, **([a+b)]**.

Problema 9.7 – Médio

Defina um novo método `void CFilaInteiros::compactaFila(int a, int b)` que remova da fila todos os valores `v` não contidos num determinado intervalo de valores `a ≤ v ≤ b`. Este intervalo é passado como parâmetro. Por exemplo, se a fila for inicialmente **4→7→3→2→5→6→8→1→nullptr** e o método for chamado com os parâmetros `compactaFila(5,8)`, então, após a execução deste método, a fila passará a ser: **7→5→6→8→nullptr**.

Obs.: pode resolver este problema chamando os métodos `retiraDaFila()` e `insereNaFila()`. Alternativamente, pode tratar a fila como uma lista ligada e manipular diretamente os seus nós.

Problema 9.8 – Médio

Defina novo método para inverter a ordem dos `N` primeiros elementos de uma fila de números inteiros. Por exemplo, se a fila contiver os valores **4 → 7 → 8 → 11 → 15 → nullptr** e `N` tiver o valor **3**, então, como resultado da execução do método, a fila deverá ficar igual a **8 → 7 → 4 → 11 → 15 → nullptr**

Obs.: pode resolver este problema chamando os métodos `retiraDaFila()`, `insereNaFila()` e utilizar uma pilha auxiliar para inverter os elementos. Alternativamente, pode tratar a fila como uma lista ligada e manipular diretamente os seus nós.

Problema 9.9 – Médio

Adicione e defina métodos na classe `CPilhaInteiros` para sobrecarregar os operadores de comparação `<, >, <=, >=` que comparam duas pilhas em termos do número de elementos nelas contidos. Isto é, uma pilha é maior do que outra se o número de elementos que tiver for superior ao da outra pilha. Tenha em atenção a declaração *standard* de cada operador.

Problema 9.10 – Difícil

Defina o método `CFilaInteiros::operator+` (operador adição) que funde duas filas numa só. A nova fila a devolver deverá ser constituída alternadamente por elementos de cada uma das filas de origem. Tenha em atenção a declaração *standard* do operador.

Problema 9.11 – Difícil

Defina um método na classe `CPilhaInteiros` para reordenar os elementos da pilha de forma que no topo fiquem os elementos pares e no fim da pilha os elementos ímpares. A ordem relativa dos números pares e ímpares deve permanecer a mesma. Considere o seguinte exemplo:
Se a pilha contiver **3 → 5 → 4 → 17 → 6 → 83 → 1 → 84 → 16 → 37 → nullptr**, então, após a execução do método pretendido, a pilha passará a ser: **4 → 6 → 84 → 16 → 3 → 5 → 17 → 83 → 1 → 37 → nullptr**.

Problema 9.12 – Médio

Defina o método `CFilaInteiros::operator-` (operador subtração) que devolve uma fila que resulta de se retirar de uma fila (o 1º operando) os elementos que também estejam contidos numa segunda fila (2º operando). Tenha em atenção a declaração *standard* do operador.

Problema 9.13 – Difícil

Adicione um novo atributo `prioridade` na classe `CNoFila` e redefina a implementação de todos os métodos da classe `CFilaInteiros`, e em particular do método `insereNaFila()` de forma que a fila respeite sempre a regra: um novo elemento é inserido no final da fila, mas à frente de todos os elementos de prioridade menor e atrás de todos os elementos com prioridade igual ou superior; no limite, se todos os elementos atualmente existentes na fila tiverem prioridade menor que o novo elemento, o novo elemento é inserido no início da fila.
Defina ainda o método `bool CFilaInteiros::setPrioridade(int N, int p)` que mude para `p` o valor da prioridade do `N`-ésimo elemento da fila a contar do início da fila e que, em consequência, posicione corretamente na fila o elemento em causa, de forma que à sua frente só existam elementos de prioridade igual ou superior e atrás só existam elementos de menor prioridade. Se o `N`-ésimo elemento não existir, o método devolve `false`, caso contrário devolve `true`.

Problema 9.14 – Médio (adaptado do exame de recurso de 05/07/2016)

Considere as estruturas de dados definidas a seguir que representam uma pilha de números reais.
a) Implemente um método para contar os números negativos existentes na pilha, utilizando uma pilha auxiliar. Não deve manipular ponteiros.
b) Implemente novamente o método acima, manipulando ponteiros e não utilizando os métodos `push` e `pop`.

```
class CNoPilha{public: double dados;   CNoPilha *proximo; };

class CPilha{
   CNoPilha *topo;   // ponteiro para o topo da pilha
  public:
   CPilha();               // já definido (elsewhere)
   ~CPilha();              // já definido (elsewhere)
   void push(double dado); // já definido (elsewhere)
   bool pop(double &dado); // já definido (elsewhere)
};
```

Problema 9.15 – Médio (adaptado do exame de recurso de 5/07/2016)

Pretende-se implementar uma aplicação informática para gerir o sistema de apoio a clientes numa determinada empresa. Para isso decidiu-se criar uma lista ligada denominada `CListaHelpdesk` que armazena a informação relativa a todas as perguntas mais frequentes que são feitas pelos clientes. Cada elemento da lista, `CNoLista`, armazena pelo menos a seguinte informação: código da pergunta (5 caracteres), resposta à pergunta (300 caracteres) e o número de vezes que a pergunta foi pesquisada (inteiro).

a) Defina as classes `CListaHelpdesk` e `CNoLista` que permitem armazenar a lista das perguntas mais frequentes.

b) Para diminuir o tempo gasto em localizar uma determinada pergunta na lista, de cada vez que é feito um acesso a um nó (para consulta ou alteração de um dos seus campos de informação), este nó é movido para o início da lista. Deste modo, os nós acedidos com maior frequência ficarão localizados no início da lista. Implemente um método `PesquisaLista` que permita procurar uma pergunta na lista (dado o seu código) e devolver a resposta associada a essa pergunta. Pretende-se também que no caso de a pesquisa ter sucesso (o código ter sido encontrado), se mova o nó correspondente para o início da lista e se incremente o número de vezes que a pergunta foi pesquisada.

Problema 9.16 – Médio (adaptado do exame de recurso de 05/07/2016)

Considere as estruturas de dados definidas a seguir e que representam uma fila de países classificados através de um *rating* que representa o seu nível de desenvolvimento económico (um número inteiro entre 1 e 10). Implemente o método `void inserePais(char *Pais, int Rating)` que insere na fila um novo país, o mais perto possível do fim da fila, mas garantindo que o novo nó fica à frente de todos os nós com um rating maior e atrás de todos os nós com um rating igual ou inferior. Note que, no limite, se a fila estiver vazia, ou se os atuais nós da fila tiverem todos um rating superior ao do novo nó, este deve ser inserido no início da fila.

```
class CNoFila{    // Nó da fila
public:
  char pais[50];
  int  ranking;           //valor mais baixo indica um país mais desenvolvido
  CNoFila *proximo; //próximo nó
};

class CFilaRankings{  // Fila contendo países ordenados pelo seu ranking
  CNoFila *inicio;    // Ponteiro para o início da Fila
  CNoFila *fim;         // Ponteiro para o fim da Fila
public:
  CFilaRankings(){inicio=nullptr;   fim=nullptr;}
  ~CFilaRankings();
  ...
};
```

Considere o seguinte exemplo:

(Canada,1) → (Alemanha,1) → (Finlandia,2) → (Austria,2) → (Belgica,3) → (França,3) → nullptr

Problema 9.17 – Médio (adaptado do teste de frequência de 16/06/2017)

Considere a classe `CFilaInteiros`, para representar filas que armazenam números inteiros, objeto de estudo nesta ficha das aulas práticas.

a) Defina um novo método constante da classe chamado `seek(int i)`, com visibilidade `private`, que devolve um ponteiro para o i-ésimo elemento da fila (a contar do início da fila). Assuma que o elemento com índice 1 é o que está no início da fila. Se o i-ésimo nó da fila não existir, o método devolverá `nullptr`.

b) Defina um novo método `public` da classe chamado `passaParaFim(int i)` que move o i-ésimo nó da fila para o fim da fila. O método não deve criar novos nós ou eliminar nós existentes; deve apenas religar a lista ligada (manipular ponteiros) para se obter o resultado pretendido. Se o i-ésimo nó da fila não existir, o método não fará nada e devolverá false, caso contrário devolverá `true`. Por exemplo, se a fila for `4, -1, 2, 8, 5` e chamar o método com o índice `i=3`, a fila fica `4, -1, 8, 5, 2`.

c) Defina um novo método `public` da classe chamado `passaParaInicio(int i)` que move o i-ésimo nó da fila para o início da fila. O método não deve criar novos nós ou eliminar nós existentes; deve apenas religar a lista ligada (manipular ponteiros) para se obter o resultado pretendido. Se o i-ésimo nó da fila não existir, o método não fará nada e devolverá false, caso contrário devolverá `true`. Por exemplo, se a fila for `4, -1, 2, 8, 5` e chamar o método com o índice `i=3`, a fila fica `2, 4, -1, 8, 5`.

Sugestão: Conceba uma solução baseada na chamada ao método da alínea a) para obter um ponteiro para o i-ésimo elemento menos 1. Repare que desta forma, tem facilmente acesso ao i-ésimo elemento da fila e ao nó anterior a este.

Problema 9.18 – Médio (adaptado do exame de recurso de 03/07/2017)

Considere a classe `CPilhaInteiros`, para representar pilhas que armazenam números inteiros, que são objeto de estudo nesta ficha das aulas práticas.

a) Defina um novo método constante da classe chamado `ponteiroPara(int i)`, com visibilidade `private`, que devolve um ponteiro para o elemento de índice i da pilha. Assuma que o elemento com índice `0` é o que está no topo da pilha. Se o nó de índice `i` da pilha não existir, o método devolve `nullptr`.

b) Defina um novo método da classe que realize a sobrecarga do operador `<<`. O operador elimina os últimos nós armazenados na pilha, a partir do nó de índice i (parâmetro `int`). O método não faz nada se `i` for menor que `0` ou maior ou igual ao número de nós da pilha, e elimina todos os nós da pilha se `i` for igual a `0`.

Sugestão: Conceba uma solução baseada na chamada ao método da alínea a) para obter um ponteiro para o i-ésimo elemento menos 1. Repare que, desta forma, tem facilmente acesso ao i-ésimo elemento da pilha e ao nó anterior a este.

Problema 9.19 – Médio (adaptado do teste de 16/05/2018)

Considere a classe `CFilaInteiros` para representar filas que armazenam números inteiros, que são objeto de estudo nesta ficha das aulas práticas. Considere agora uma classe derivada da anterior, `CListaCircular`, que inclui mais um atributo (atual, privado) que é um ponteiro para o nó corrente (inicialmente o primeiro). Esta classe tem acesso a todos os atributos e métodos da classe mãe (ou classe base). Existem os métodos (públicos) da classe mãe e ainda o método `getNext()` que devolve o valor do próximo elemento (a seguir ao corrente, que passa a ser o novo nó corrente). O nó a seguir ao último (fim da fila) considera-se que é o primeiro (início da fila).

a) Escreva a declaração da classe de objetos `CListaCircular`. Nesta alínea, não deve definir/implementar qq. método *inline*.
b) Implemente o método `getNext()`.
c) Implemente uma função `main()` para teste que cria uma lista circular com os números 1 a 10 e mostra os elementos obtidos com 15 chamadas sucessivas a `getNext()`.

Problema 9.20 – Médio (adaptado do teste de 15/06/2018)

Considere as classes `CFilaInteiros` e `CPilhaInteiros` para representar filas e pilhas que armazenam números inteiros, que são objeto de estudo nesta ficha das aulas práticas.

a) Defina um novo construtor da classe `CFilaInteiros` que aceite como parâmetro uma pilha de inteiros e que construa uma fila com os números contidos na pilha. O número do topo da pilha deve ficar no início da fila e o de baixo, no fim.
b) Defina um novo construtor que aceite como parâmetros uma pilha de inteiros e um booleano e que construa uma fila com os números contidos na pilha. Se o booleano for `true`, só são inseridos na fila os elementos pares da pilha; se for `false`, só os ímpares.

Nota: No final, a pilha pode ficar vazia. As duas classes são independentes (nenhuma é amiga da outra).